**AFRL-IF-RS-TR-2004-169**
**Final Technical Report**
**June 2004**


# CONTROL OF AGENT BASED SYSTEMS (CoABS) GRID

**Global InfoTek, Inc.**

**Sponsored by**
**Defense Advanced Research Projects Agency**
**DARPA Order No. J386**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*



**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**
**ROME RESEARCH SITE**
**ROME, NEW YORK**

**STINFO FINAL REPORT**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-169 has been reviewed and is approved for publication.

APPROVED: /s/
WAYNE A. BOSCO
Project Engineer

FOR THE DIRECTOR: /s/
JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 074-0188*



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | June 2004 | **FINAL** **Sep 98 – Mar 04** |

**4. TITLE AND SUBTITLE**

CONTROL OF AGENT BASED SYSTEMS (CoABS) GRID

**5. FUNDING NUMBERS**

C - F30602-98-C-0267
PE - 62702F, 63728F, 63706E
PR - AGEN
TA - T0
WU - 14

**6. AUTHOR(S)**

Dennis E. Brake
Gholamreza Emami

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Global InfoTek, Inc.
Suite 200
1920 Association Drive
Reston VA 20191

**8. PERFORMING ORGANIZATION REPORT NUMBER**

N/A

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlingon VA 22203-1714

AFRL/IFTB
525 Brooks Road
Rome NY 13441-4505

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

AFRLIF-RS-TR-2004-169

**11. SUPPLEMENTARY NOTES**

AFRL Project Engineer: Wayne Bosco/IFTB/(315) 330-3578 Wayne.Bosco@rl.af.mil

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** ***(Maximum 200 Words)***

The Control of Agent Based Systems (CoABS) Grid provides the middleware that enables dynamic interoperability of distributed, heterogeneous, objects, services, and multi-agent systems. It has been used to produce militarily relevant technical integration experiments where legacy systems and multi-agent systems were integrated to solve real-world problems. The CoABS Grid features flexible run-time communications and dynamic registration and discovery of relevant participants. It is adaptive and robust, with the system evolving to meet changing requirements without reconfiguring the network.

The CoABS Grid integrated heterogeneous agent-based systems, object-based applications, and legacy systems. It includes a method-based application programming interface to register agents, advertise their capabilities, discover agents based on their capabilities, and send messages between agents. The Grid also provides a logging service, a security service, and remote event notifications.

**14. SUBJECT TERMS**

Agent Based Computing, Multi-agent Systems, Middleware, Dynamic Interoperability, Legacy Systems.

**15. NUMBER OF PAGES** 17

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

# Table Of Contents

# 1 Introduction

Control of Agent-Based Systems (CoABS) is a program of the U.S. Defense Advanced Research Projects Agency (DARPA) to develop and demonstrate techniques to safely control, coordinate, and manage large systems of autonomous software agents. CoABS project investigated the use of agent technology to improve military command, control, communication, and intelligence gathering. Global InfoTek Inc. (GITI) is pleased to present an overview of our work in developing the CoABS Grid. The CoABS Grid is the plumbing that connects the components developed by CoABS researchers to solve real world problems.

# 2 CoABS Vision

The military environment is dynamic with quickly changing operations, hardware and software moving, connecting and disconnecting, and bursty bandwidth availability. Inflexible stove-piped legacy systems that were never meant to be integrated are, nevertheless, of vital importance to military planning and operations. Multiple hardware and software platforms as well as data interfaces and standards further complicate the picture. In addition, military personnel are overwhelmed by the increased data availability from the modern battlefield and suffer from information overload with no adequate tools to filter and correlate the data. A goal of CoABS is to enhance the dynamic connection and operation of military planning, command, execution, and combat support systems to quickly respond to a changing operational picture. Software agents are being developed to work side-by-side with human military planners and operators to ease the burden of their daily tasks. Critical technical issues associated with scalable, robust, and adaptive coordination and control strategies that allow large teams of heterogeneous software agents to coordinate effectively are being addressed by CoABS researchers.

Broadly speaking, there are four different kinds of agents of key interest to the military: those that are aimed at complex problem-solving; those that find, filter and present information for users; those that provide services to other agents to help them cooperatively solve complex problems; and those that provide translational services between agents using different standards, communications protocols, languages, etc. Large-scale, cooperative teams, comprised of interacting agents from all four groups, could offer new capabilities that are now beyond the realm of software designers. An infrastructure that could provide these capabilities would allow software developers to design smaller pieces of code that would primarily function on solving problems via interaction with each other, rather than by each trying to duplicate functions provided by others.

In such a world, heterogeneous systems, separately developed, could be integrated into compound systems at run-time, based on the needs of the particular problems being solved. Finding these code pieces would be enabled by yellow page servers using taxonomies of common functionalities. Where gaps might exist between the agents, functions such as translation services could provide greater interoperability by seamlessly filling in the pieces. Users would be able to program and interact with their agents using interface tools that would allow them to set preferences and describe needs without needing to specify algorithmic details. In addition, the entire "grid" of cooperating agents could be managed by the brokering agents – which would help to manage the efficient flow of information across the grid. These agents could also provide tools for access control and information security, and they could provide a database allowing post hoc analysis of problem solving and other grid management services.

Such a system would provide many capabilities that cannot be realized with any but the most state-of-the-art of today's tools. Users would be able to setup queries to search and filter large knowledge

bases, to search through the net or other information sources, or to find computational resources needed for the problems they were trying to solve – all without needing to know the details of the underlying systems or information repositories. Current "legacy" systems could be brought to the grid through software wrappers and service descriptions, allowing their functionality to be tapped without major recoding. In addition, the cooperative nature of the problem solving, using existing software components, would allow both military and industrial users to develop large scale applications without large scale software development efforts.

This is the vision of the software-of-the-future that drives the Control of Agent-Based Systems (CoABS) program being supported by the Defense Advanced Research Projects Agency (DARPA). CoABS has provided funding to almost two dozen universities, companies, and research institutes to cooperate in developing the underlying science for developing and controlling such an agent grid. In addition, CoABS is developing software components that can be used to validate this concept. By a process of iterative development and evaluation, the CoABS researchers are providing the necessary software tools and techniques to bring this vision into being.

## 3 CoABS Grid

The CoABS Grid provides the middleware that enables dynamic interoperability of distributed, heterogeneous, objects, services, and multi-agent systems. It has been used to produce militarily relevant technical integration experiments where legacy systems and multi-agent systems developed by CoABS researchers were integrated to solve real-world problems. The CoABS Grid features flexible run-time communications and dynamic registration and discovery of relevant participants. It is adaptive and robust, with the system evolving to meet changing requirements without reconfiguring the network. A significant effort is being made to transition and deploy the CoABS Grid and associated agent technologies to the operational user community.

The CoABS Grid integrates heterogeneous agent-based systems, object-based applications, and legacy systems. It includes a method-based application programming interface to register agents, advertise their capabilities, discover agents based on their capabilities, and send messages between agents. The Grid also provides a logging service, to log both message traffic and other information; a security service to provide authentication, encryption, and secure communication; and event notification when agents register, deregister, or change their advertised attributes. These capabilities can be summarized as:

- A foundation to easily wrap legacy systems
- A network for distributed discovery and runtime integration
- Registration, discovery, advertise, and search functions
- An agent communication infrastructure
- Event notification for registration / modifications
- Automatic logging of agent messages
- Message encryption and service authentication
- Administration and visualization tools
- Publish and subscribe services
- Operation through firewalls

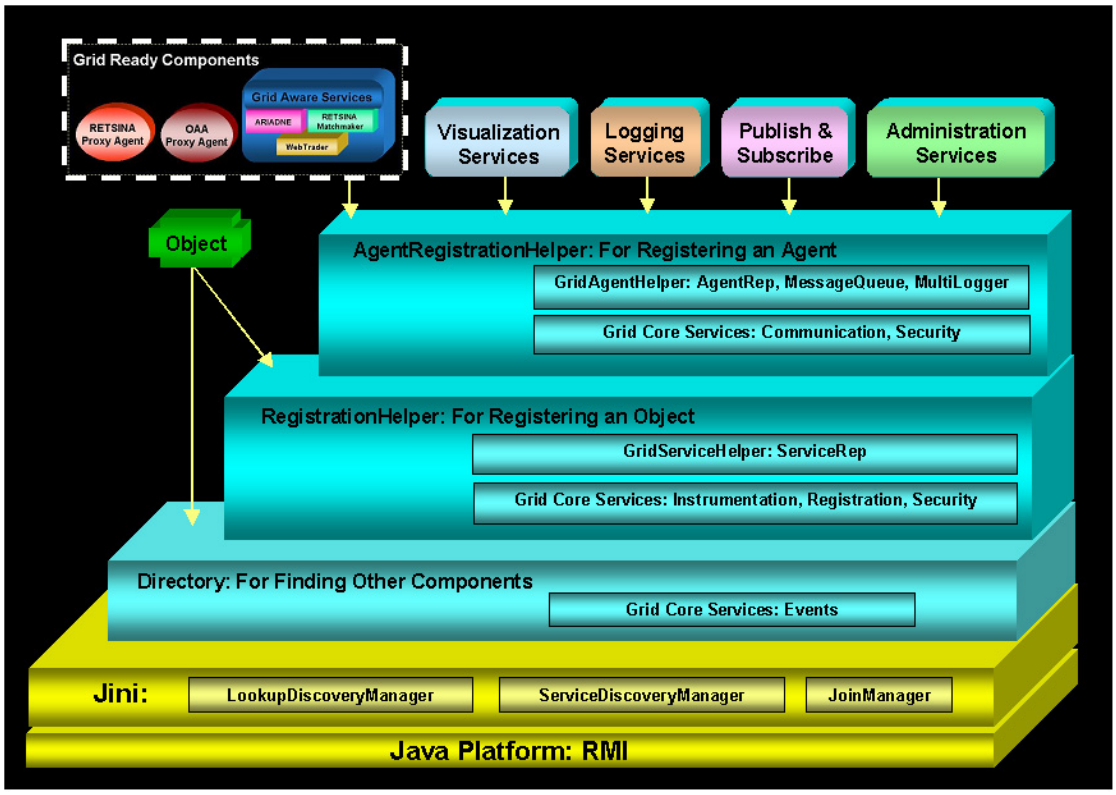
## 3.1 Grid Architecture

The CoABS Grid supports the development of applications for dynamic domains such as military command and control, which require the composability, adaptability, and autonomy provided by software agents interoperating in dynamic, mixed-initiative teams with human users.

Using the Grid, agents and wrapped legacy systems can (1) describe their needs, capabilities and interfaces to other agents and legacy systems; (2) find and work with other (lightweight) agent components and legacy systems to accomplish complex tasks in flexible teams, versus a single monolithic application; (3) interact with humans and other agents to accept tasking and present results, and (4) adapt to changes in the battlespace, the task at hand, or the computing environment.

The Grid does this by providing access to shared protocols and ontologies, mechanisms for describing agents' capabilities and needs, and services that support interoperability among agents and legacy systems at flexible levels of semantics—all distributed across a network infrastructure.

Although many architectures provide some of the interoperability and other services that the Grid provides, each architecture typically supports specialized constructs, communication, and control mechanisms. This specialization is desirable because particular systems can use mechanisms appropriate to the problem domain/task to be solved. A layered view of the architecture is shown in the following figure.




The Grid is not intended to replace current architectures but rather to augment their capabilities with services supporting trans-architecture teams. Agent technologies support semantically rich conversations among these agents (and wrapped legacy systems), which allow them to interoperate outside their local "community." An analogue is the Internet's bridging of heterogeneous networks by gateways and protocols. Programmers will make their components "Grid Aware," much as many network applications are now made "Internet ready" or "Web ready" by supporting protocols and

languages such as TCP/IP, HTTP, HTML, and XML. Furthermore, programmers will want to make their components "Grid Aware" to enable them to participate in dynamic teams that leverage other components discovered at runtime.

The Grid leverages emergent agent technology and standards from the CoABS community and industry. Not every component needs to be an agent. Thus, the Grid also leverages other technologies supporting component interconnectivity and interoperability among objects and other components (e.g., OMG's CORBA, Sun's Jini).

The CoABS Grid includes a method-based application programming interface to Grid services. All communication is point-to-point. For this reason, it scales to a large number of agents with no restrictions beyond those imposed by network bandwidth. Agent registration and discovery, on the other hand, are reliant on one or more lookup services. Scalability experiments are described in the following section. These experiments showed no degradation in performance with up to 10,000 agents.

### 3.2 Jini™ Concepts Used by the Grid

The Grid is built using Jini™ Network Technology developed by Sun Microsystems. Jini™ is a specification for dynamic service discovery. The Jini™ Technology Starter Kit that contains a contributed implementation of the specification can be downloaded for free from the Sun website. Commercial versions of some Jini™ components are also available. The robust and dynamic nature of the CoABS Grid is derived from Jini™. The Grid provides helper utility classes that are local to an agent and that shield the Grid user from the complexity of Jini™. It also provides services to facilitate agent communication and logging. The CoABS Grid takes advantage of three important components of Jini™:

1) the concept of a service, which is used to represent an agent,
2) the Lookup Service, which is used to register and discover agents and
3) Entries, which are used to advertise an agent's capabilities.

A Jini™ service is a Java object that is serialized and stored in the Lookup Service (LUS). The LUS supports lookup of services based on type, attribute values, and unique identifier. When a client performs a lookup through the LUS, the service object is returned to the client. The service may optionally be a proxy that uses a remote connection to communicate back to the true service at a different location. The remote connection is transparent to the client and can use any transport protocol (e.g. RMI, CORBA, or secure sockets).

The LUS grants leases to registered services, assigns globally unique identifiers to services, and supports lookup of services. It is the service's responsibility to maintain its lease with the LUS; however helper classes to do this automatically. If a service can't maintain its lease because of either failure of the service or failure of the network connection between the service and the LUS, the service will be purged from the LUS, so that the LUS contents remain current.

Jini™ provides helper classes that use a multicast discovery protocol to find any LUSs that are running within a local area network. No prior knowledge of the machine name or port that the LUS is running on is required. Jini™ also provides a unicast discovery protocol to find LUSs outside the local area network. Service registration is maintained in all local and distant LUSs. The registration is automatically propagated to any new LUS processes that are started. Multiple LUSs can be run for robustness and scalability. If one goes down, the others will still maintain registration and lookup.

Jini™ services are described in the form of a Jini™ Entry. An Entry is a collection of service attributes that is stored in the LUS along with the service. Many Entries can be stored for a single service. An Entry is an object that has public fields, cannot contain primitive types, and has a no-parameter

constructor. Any Serializable object that meets these criteria can be an Entry as long as it implements the marker Jini™ Entry interface. Entry templates are used in the Grid lookup methods to find registered services.

### 3.3 CoABS Grid Extensions

The Grid provides helper utility classes that are local to an agent and that hide the complexity of Jini™. These classes automatically find any LUS in both the local area network and user-designated distant machines. The Grid supports agent and service discovery based on Jini™ Entries and arbitrary predicates as well as by service type. The Grid also provides event notification when agents register, deregister, or change their advertised attributes.

The Grid defines a Jini™ service interface called the AgentRep, which is a proxy to the agent. This interface defines a method called addMessage(), which uses a remote connection to deliver a message back to the agent. Thus, when a client agent calls a Grid lookup method, a proxy that allows immediate direct communication back to the agent is returned. The client agent can include its own AgentRep proxy in the message it delivers, so that two-way communication can be established with no further lookup. The Grid is transport neutral in terms of agent communication. The Grid defines the interface, but the agent proxy is free to use any transport.

The Grid currently provides a DefaultAgentRep implementation that uses RMI for message transport. A DefaultAgentRep downloaded to a client is connected to a MessageQueue object local to the agent using RMI. A MessageListener interface is also defined to allow agents automatic notification of incoming messages. Several classes of CoABS Grid messages are provided. Some include text messages only, while others allow data attachments. The Grid is language neutral; any agent communication language can be used.

Agent communication is fully distributed, in that each agent sending a message communicates directly with the receiver, using the proxy registered by the receiver. The sender is unaware of the transport mechanism being used, although currently RMI is the default.

## 4 CoABS Grid Experiments

GITI has developed the Grid and is working to ensure its ability to perform up to expectations. The Grid has already been used successfully in a series of naval Fleet Battle Exercises (FBEs), by the Joint Inter-Agency Task Force-East, by the Army Communications and Electronics Command, by the Air Mobility Command, by the Expeditionary Pervasive Sensing (ESG) Enabling Experiments (EEE), and in the Air Force's Joint Battlespace Infosphere project.

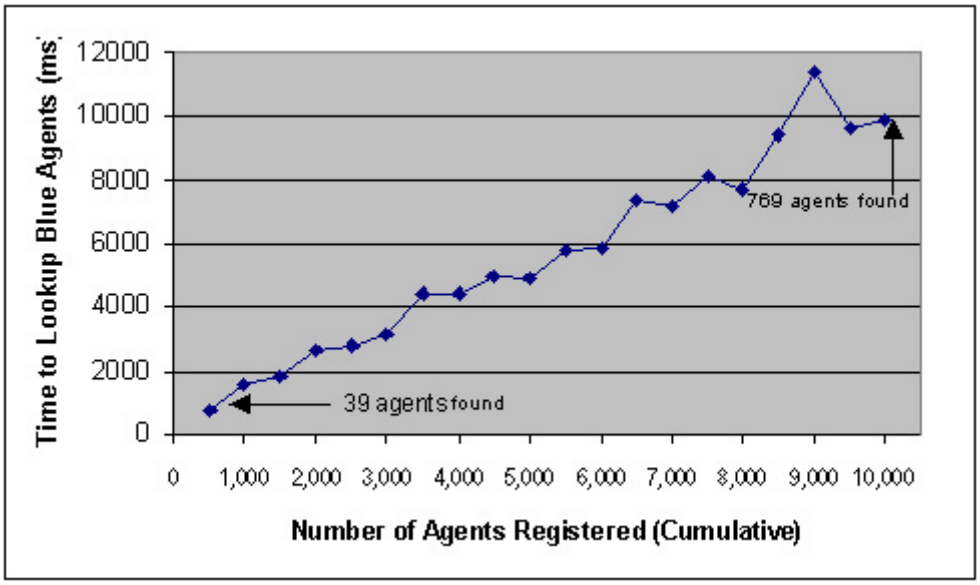
As the CoABS Grid has been used in more realistic military experiments, such as US Navy Fleet Battle Experiments and the Coalition Agents Experiment (CoAX) involving US, British, and Australian participants, it has become important to investigate the scalability of the CoABS Grid infrastructure. Several sets of experiments have been conducted.

The first set of experiments were conducted in August 2000 and investigated how lookup times scale as the lookup service becomes more populated. The experiments were not designed to give a definitive quantitative measure to the scalability of registration and lookup of the CoABS Grid. Instead, they were designed to give a qualitative understanding of whether performance problems become apparent with a highly populated LUS. As such, while the experiments were performed under conditions of light network load, no effort was made to completely isolate the network for the purposes of the experiments. Also, no effort was made to create a testbed of identical machines on which to run the various components of the experiments. In addition, the measurements were made on the machines

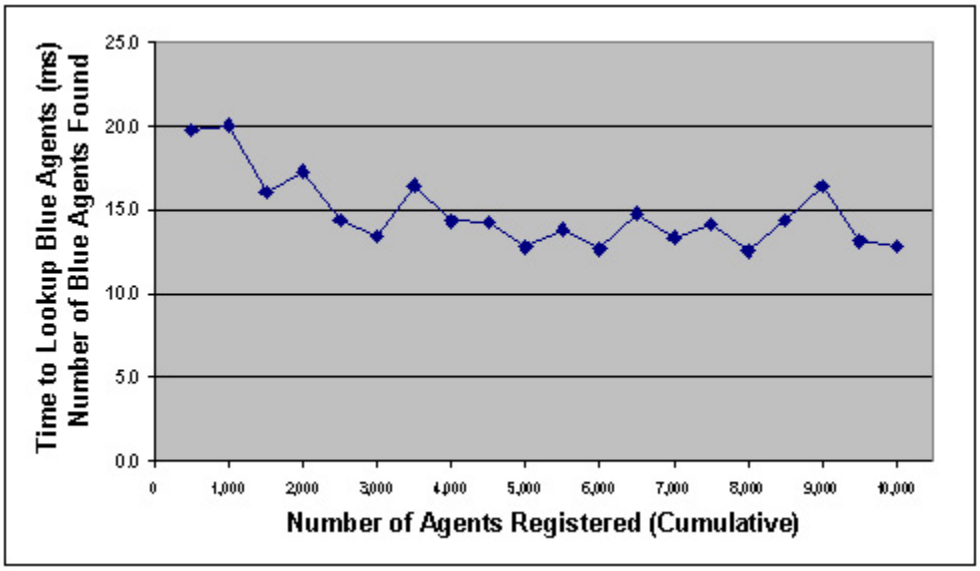
where the agents were created, rather than the machine the lookup agent was running on, so network latency and client machine speed affect the results. The timing numbers shown here are to be regarded as indications of performance trends, not as definitive indications of the time it takes to perform various functions.

In these experiments, 500 agents were registered at a time with the CoABS Grid, until a total of 10,000 agents were registered. After each group of 500 agents was registered, twenty-two different agent queries were performed measuring how long it took to fetch the agents and the number of agents retrieved. Sequential lookup queries scaled well to 10,000 agents. Lookups that retrieved a single agent, as well as lookups that matched no agents were not affected by the number of agents registered. Experiments performed by an independent subcontractor also found constant lookup times for lookups that returned 120 agents with a maximum of 600 agents registered, and lookups that returned 250 agents with a maximum of 2500 agents registered. Time to lookup multiple agents increases proportionally to the number of agents retrieved. Time to lookup multiple agents is independent of the number of agents registered.

The results of these experiments were presented at the Autonomous Agents 2001 Conference, Second International Workshop on Infrastructure for Agents, Multiple Agent Systems (MAS), and Scalable MAS on May 29, 2001. Two of the graphs from that paper are shown here:

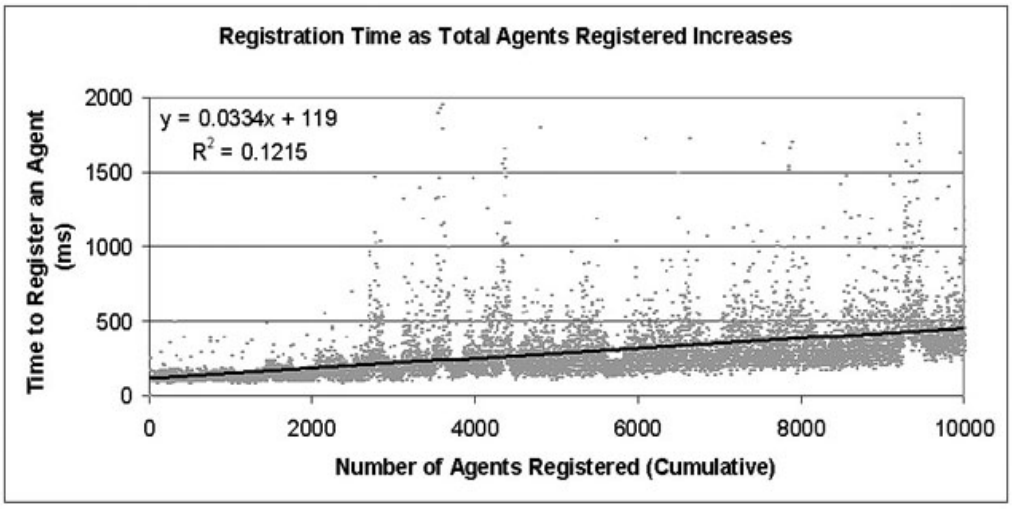



Lookup Time vs. Number of Agents Registered




Lookup Time Divided by Number of Agents Retrieved

A second set of experiments were run with a newer version of the Grid code in April of 2001. These experiments repeated the lookup experiments done previously. Measurements of how long it took to register the agents, as the lookup service became more populated were also collected. A paper describing these results was published in "Autonomous Agents and Multi-Agent Systems" on Infrastructure and Scalability. The following figure shows the results of the registration experiment:




Registration Time Divided vs. Number of Agents

## 5 Grid Design

The Grid was designed to meet several goals. It must be simple to use for the average user, yet still provide full flexibility for users that needed to use the Grid in less common ways. The design must be

efficient, with classes shared where possible to minimize both instance creation and Jini™ processing. In addition, Grid classes must be easily extendable, to simplify wrapping of legacy systems.

The Grid design makes a distinction between a service and an agent, in that an agent is considered to use a message-passing paradigm of communication, while a service uses a method-based mechanism. Although this is not a universal distinction within the agent community, it assists in the organization of Grid functionality. A primary goal of the Grid is to integrate agent, object, and legacy systems, so being able to accommodate both communication styles is built-in.

A basic tenant of Jini™ is that communities agree on well-known interfaces for the services they use. One of the main contributions of the CoABS Grid is a well-known interface to support agent-to-agent communication called the AgentRep. The AgentRep has a single addMessage() method that takes a Message as its argument. The DefaultAgentRep implementation provided with the Grid uses a MessageQueue and RMI to support the agent, but the interface only requires the single method. The MessageQueue is used to hole the Messaes in the order that they were received.

The Message interface is implemented by the BasicMessage. Its receiver property contains the string name of the intended Message receiver, the sender filed contains an AgentRep to allow a reply, the rawText field holds the contents of the message, and an Agent Communication Language (ACL) field describes the agent communication language being used.

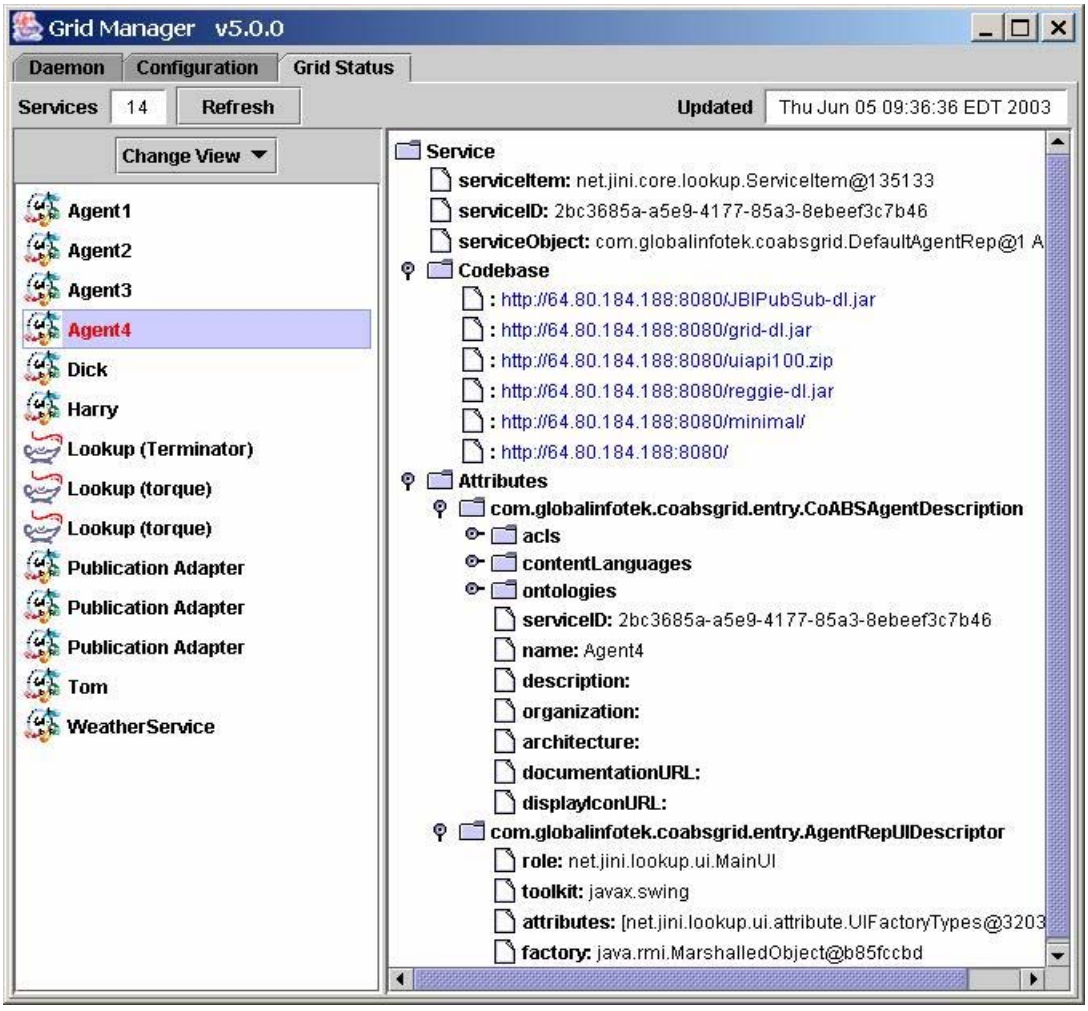
A DataMessage extends the BasicMessage and can contain any Serializable data. It can be used to hold simple sequence numbers, Strings, or even code continuations that resume calculations when they are received.

# 6 Grid Components

The Grid is comprised of many cooperating components that can be used individually or in concert. The following sections give an overview of the major components that are included in the Grid distribution.

## 6.1 The Grid Manager

The GridManager is a GUI that is used to control the underlying processes that need to be run to support the Grid. The GridManager is composed of three panels, labeled Daemon, Configuration, and Grid Status. The Daemon panel supports starting the background daemon processes, the Configuration panel supports modifying the run-time configuration properties, and the Grid Status panel shows a dynamic list of registered agents and services. The following figure is an example display of the Grid Status panel.




## 6.2 Grid Logging Service

The Grid Logging Service allows agent messages to be saved, monitored, and replayed. The Logging service is comprised of three components: the Logger, the Log Viewer, and The Agent Sequence Visualizer.

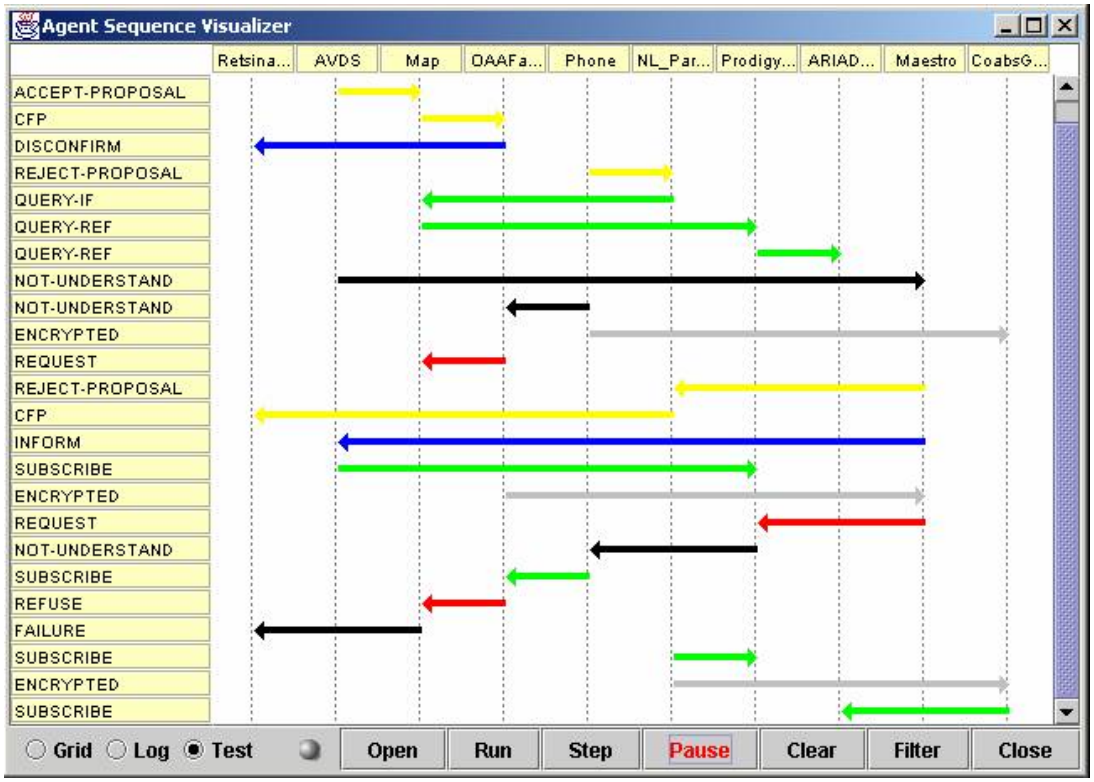
As each agent starts up it will attempt to find a Logger registered with the lookup service. Multiple Loggers may be running. Any logger that has been found will record all agent message traffic in an XML data file. Agents and services may also record other special events with the logger by using the logging API. By default, agents communicate with the logger using the FIPA agent communications language.

The contents of the log file can be viewed using the Log Viewer. The Log Viewer application is a Java Swing application that runs as a Grid agent. The Log Viewer can open and display the log file, which is an XML document. The Log Viewer GUI displays each log entry and allows drill down for more detailed information.

The Agent Sequence Visualizer (ASV) is a Java component that can be used to observe, debug, and demonstrate agent activity. It is implemented as a CoABS Grid agent, and it supports live monitoring

of other agents, as well as replay of previously logged Grid sessions. No modification of existing agents or services is required. In particular, agents do not need to send messages to the ASV (or make calls on it) since the Grid's logging system provides this functionality.

The ASV uses dynamic UML-type sequence diagrams to represent agent activity. These diagrams show inter-agent message traffic over time. A resizable, scrolling main window displays a complete session history, which may include a large number of agents and messages. The following figure contains a screen capture of this window.




## 6.3 AutoGenerator

One of the main benefits of using the Grid is the ease of integrating legacy systems. This can be done either manually or using the CoABS Grid AutoGenerator. The CoABS Grid AutoGenerator is a wizard-based tool that enables wrapping legacy code as a Grid service or a Grid agent.

The AutoGenerator creates the necessary Java code to expose the legacy code on the Grid as a service or agent. It can also generate Service User Interface classes to allow an existing GUI for the legacy code to be associated as a ServiceUI Jini Entry to the service or agent. It can also generate Ant scripts for compilation and execution of the generated service or agent (provided that the GUI is Serializable and is presented by a JFrame or JComponent). The AutoGenerator has a number of screens that enable user code to be inserted directly into the generated code, in most cases eliminating the need to edit any of the generated Java code. For the simplest of services and agents, the user can accept any defaults and no Java code at all needs to be manually written.

## 6.4 Grid Security Service

There are three components to the GridSecurityService: authentication, encryption, and communication. The authentication service is used to digitally sign and validate a ServiceRep. The

encryption service provides a way to encrypt and decrypt any Serializable object. The communication service uses secure sockets to send messages to an AgentRep's message queue.

The GridSecurityService uses the JDK Security API and the reference security provider implementation provided by Sun. However, the GridSecurityService is not dependent upon Sun's implementation and will work with any properly installed service provider. By default the GridSecurityService uses X509 certificates, a Java Keystore (JKS), and generates signatures using the SHA1 hash algorithm and the DSA encryption algorithm. Object encryption uses the Data Encryption Standard (DES) algorithm provided in the JDK and the communications tools use secure sockets (SSL). These tools may be utilized together or individually depending upon user's requirements.

The authentication services consist of the public methods *signProxy*, *verifyProxy*, *getCertificateChain*, and *checkTrust*. The *signProxy* method will accept a service proxy object and return a SignatureEntry. This SignatureEntry can be added to the advertised capabilities of the service using the RegistrationHelper class. When clients locate the service they can use the SignatureEntry to verify the signature of the service and decide whether or not to use the proxy.

The encryption services consist of generateSymmetricKey, initCipher, encrypt, and decrypt methods. The generateSymmetricKey method will create a key that can be used to encrypt an Object. The initCipher method must be called before either the encrypt or decrypt methods are used. The initCipher method initializes the Cipher object with the key and algorithm that is to be used for encryption. The encrypt and decrypt methods are both overloaded to accept either a String or any Serializable Object.

Secure Communication uses RMI over SSL to provide over-the-wire encryption of message information. Agent designers have complete control over how messages are sent to their message queues. By using a SecureMessageQueue instead of the default MessageQueue, the agent can require that clients use SSL to send messages to the queue.

A Secure logger is implemented in the SecureLogProxyAgent class and provides a good example of all the security tools described in the previous sections. The secure logger is a regular Grid service; it registers with the Grid and waits to receive messages for logging. To insure secure communication it uses a SecureMessageQueue for over-the-wire encryption of message information. Once a message has been received, the secure logger encrypts the raw text and writes the encrypted message to the log file. The SecureLogDecryptor is a java application that can be used to read an encrypted log file and decrypt the message content. This tool reads the log file and displays the decrypted information on the command line.

## 6.5 Publish and Subscribe Service

Publish and Subscribe is a Grid add-on service that is part of Joint Battlespace Infosphere (JBI) and is provided by the Air Force Research Laboratory/Information Directorate (AFRL/IF) at the Rome Research Site. This service has been fully integrated with the Grid and provides a simple yet powerful paradigm for publishing objects.

Any Serializable Java object can be published. This includes regular Java Objects, Remote Objects, Proxy Objects, and data files (gif and jpeg images, word documents, etc.). Users of the Publish and Subscribe services may characterize their metadata as an XML Schema and XML document that is valid with respect to the supplied schema or, alternatively, users may characterize their metadata using arrays of Jini™ Entries as metadata descriptors. The metadata may then be used at registration time to advertise the attributes of publishable objects.

A PubAdapter class handles the publication of objects. It allows for the registration of multiple objects (via metadata descriptors) and the subsequent publication using separate publisher threads.

A SubAdapter class handles the subscriptions for publishable objects on behalf of a subscribing application. It allows for subscription registration for multiple objects (via metadata descriptors). Using the API provided, you may subscribe for the delivery of objects based solely on the metadata used (by the publisher) to characterize the publishable objects. In addition, you may specify either an *XPath* expression or an *XQL* query string at registration time to allow for the additional processing of XML documents before they are actually sent to the subscriber.

### 6.6 GridTunnel

The GridTunnel is a Grid service that enables agents and publishers to operate through firewalls. This service seamlessly tunnels agents and publishers between separate LANs. Agents and publishers registered on one LAN are automatically registered on the second LAN. The agents and publishers do not need to be modified in any way to participate in the tunneling. The GridTunnel was designed to support the following requirements:

- Support agent and publish/subscribe through multiple levels of firewalls
- Expose only a single configurable port through a firewall
- Tunnel and receive agent messages without modifying existing agents
- Support publisher and subscriber registrations and data exchange
- GridTunnels will only communicate with specific peers.
- Communications between peers will be over SSL
- SSL will mutually authenticate the peers
- Java policy file will determine all permissions granted to a GridTunnel

The GridTunnel uses a technique of wrapping (or nesting) of objects to support the remote communications. This technique demonstrates the ability of the service being registered to control how clients communicate with it.

An agent is wrapped in a TunnelAgent (subclass of AgentRep). The TunnelAgent knows how to communicate back to the GridTunnel that created it. When it communicates back to the GridTunnel, it will be unwrapped and it's inner (or nested) proxy object exposed. If this inner object is another TunnelAgent, it will communicate with its GridTunnel. Finally the process will unwrap to the original agent proxy and it will receive its Message. As part of this process the codebase must be wrapped as well as any AgentRep contained in the sender field of the Message object being sent.

Publishers are wrapped in a TunnelPublisher (subclass of PubAdapterAdminProxy). A similar process is used for wrapping a publisher proxy and codebase. Subscribers that discover a TunnelPublisher will try to register with the publisher. This registration will be propagated back to the original publisher by unwrapping the nested objects. The subscriber proxy is wrapped in a TunnelSubscriber (subclass of SubAdapterProxy). After a subscriber is registered with a publisher, the publisher will use the unwrapping process to publish objects back to the original subscriber. Codebase wrapping is also used for publishers and subscribers.

## 7 Summary

The CoABS Grid has been a very successful project. It has been requested by over 400 commercial and military users. We have worked with over two dozen universities, companies, and research institutes to cooperate in developing the underlying science for developing and controlling such an agent grid.

The Grid has been used successfully in a series of naval Fleet Battle Exercises (FBEs), by the Joint Inter-Agency Task Force-East, by the Army Communications and Electronics Command, by the Air Mobility Command, by the Expeditionary Pervasive Sensing (ESG) Enabling Experiments (EEE), and in the Air Force's Joint Battlespace Infosphere project. A significant effort is being made to transition and deploy the CoABS Grid and associated agent technologies to the operational user community.

The Grid leverages emergent agent technology and standards from the CoABS community and industry. The Grid also leverages other technologies supporting component interconnectivity and interoperability among objects and other components (e.g., OMG's CORBA, Sun's Jini, WebServices).

A number of experiments were performed to demonstrate the performance and scalability of the Grid. The results of these experiments were presented at the Autonomous Agents 2001 Conference, Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS on May 29, 2001 and also published in "Autonomous Agents and Multi-Agent Systems" on Infrastructure and Scalability.